Algorithmen & Datenstrukturen
Midterm Test 2

Martin Avanzini <martin.avanzini@uibk.ac.at>
Thomas Bauereiß <thomas.bauereiss@uibk.ac.at>
Herbert Jordan <herbert@dps.uibk.ac.at>
René Thiemann <rene.thiemann@uibk.ac.at>

14. Juni

Name	
Matrikelnummer	

Aufgabe	**mögliche Punkte**	**erreichte Punkte**
1.1	10	
1.2	5	
2.1	7	
2.2	5	
2.3	8	
3.1	7	
3.2	3	
Gesamt	**45**	

Aufgabe 1) Mengen (15 Punkte) Ein Algorithmus benötigt für seine Abarbeitung eine effiziente Datenstruktur `Set` zur Verwaltung einer Menge von natürlichen Zahlen (`int`). Folgende Operationen sollen mit der gegebenen Laufzeitkomplexität auf einer Menge A ausgeführt werden können:

- Einfügen - im Durchschnitt $\mathcal{O}(1)$
- Löschen - im Durchschnitt $\mathcal{O}(log(|A|))$
- Member-Ship Test (`contains(int x)`) - im Durchschnitt $\mathcal{O}(1)$
- Iteration über alle Elemente - im Durchschnitt $\mathcal{O}(|A|)$
- Durchschnitt zweier Mengen A und B (`intersect(Set A, Set B)`) - im Durchschnitt $\mathcal{O}(min(|A|, |B|))$

1. Wählen Sie eine geeignete Datenstruktur für die Realisierung der Menge. Beschreiben Sie alle notwendige Details. Begründen Sie, weshalb Ihre Wahl den Anforderungen (Laufzeiten) entspricht.
2. Implementieren Sie in Java-ähnlicher Syntax die Funktion `intersect(Set A, Set B)` zur Berechnung des Durchschnitts der Mengen `A` und `B`. Notwendige Operationen wie `contains(int x)`, `add(int x)`, Iteratoren, usw. können Sie als gegeben voraussetzen. Begründen Sie, dass Ihre Implementierung eine Laufzeit von $\mathcal{O}(min(|A|, |B|))$ hat.

Aufgabe 1) Lösung

Aufgabe 2) Dynamisches Programmieren (20 Punkte) Sei a ein Array über Integer-Werten der Länge $n > 0$. Eine *zusammenhängende Teilsequenz von a mit maximaler Summe* ist gegeben durch ein Paar (i, j) von Zahlen $0 \leqslant i \leqslant j < n$ sodass die Summe $\Sigma_{k=i}^{j} a[k]$ maximal ist. (Insbesondere gilt $\Sigma_{k=i'}^{j'} a[k] \leqslant \Sigma_{k=i}^{j} a[k]$ für jedes weitere Paar (i', j').)

Solch ein Paar lässt sich durch dynamisches Programmieren errechnen indem man folgende Rekursionsgleichung betrachtet:

$$\begin{aligned} U(0) &= a[0] \\ U(k+1) &= \max\{U(k) + a[k+1], a[k+1]\} \end{aligned}$$

Der Wert $U(k)$ beschreibt die maximale Summe einer zusammenhängenden Teilsequenz endend mit Index k. Eine Lösung (i, j) kann wie folgt berechnet werden:

- Der Index j wird so gewählt, dass $U(j)$ maximal ist (im Bereich $0 \leqslant j < n$).
- Zur Bestimmung des Anfangsindex i wird eine Abbildung $L\colon \{0, \ldots, n-1\} \to \mathsf{Int}$ errechnet sodass die Summe der Einträge von $a[L(j)], \ldots, a[j]$ maximal ist. Es gilt also $i = L(j)$. Beachten Sie, dass sich L anhand der Berechnung von U erstellen lässt.

1. Implementieren Sie die Berechung von U mittels einer Methode `int[] u(int a[])`.
2. Sei `a = {-1,2,3,-2}`. Bestimmen Sie die Werte $U(k)$ sowie $L(k)$ für $0 \leqslant k < 4$. Geben Sie die Lösung (i, j) an.
3. Erstellen Sie die Rekursionsgleichung für L anhand der Werte $U(k)$ und $a[k]$ $(0 \leqslant k < n)$.

Aufgabe 2) Lösung

Aufgabe 3) Topologische Sortierung (10 Punkte) Gegeben sei ein Graph $G = (V, E)$ mit

$$
\begin{aligned}
V = \{&a, b, c, d, e, f, g, h\} \\
E = \{&(a, g), \\
&(c, b), (c, d), (c, e), \\
&(d, b), \\
&(e, b), \\
&(f, a), (f, c), (f, h), \\
&(g, c), \\
&(h, g)\}
\end{aligned}
$$

1. Bestimmen Sie eine topologische Sortierung des Graphen G, indem Sie den in der Vorlesung vorgestellten effizienten Algorithmus anwenden. Geben Sie die Zwischenergebnisse an (z.B. in Form einer Tabelle) sowie die resultierenden Ordnungsziffern $ord(v)$ für alle Knoten $v \in V$.

2. Zeichnen Sie den Graphen G mit Hilfe der Ergebnisse der vorherigen Teilaufgabe. Platzieren Sie dabei die Knoten in der Reihenfolge der topologischen Sortierung entlang einer Geraden, und achten Sie darauf, dass sich die Kanten nicht überschneiden.

Aufgabe 3) Lösung